Corbin Mayes - 1/18/19

Mazeworld Assignment

A-star Search

To implement the A-star search into my program I took the given code and implemented a best-first search and then added the heuristic to be added to the transition cost for the best-first search to become an A-star search. I later altered it slightly to be usable for both my MazeworldProblem and my SensorlessProblem classes. I stuck with just using the heap for storing items and am just dealing with computational problems cause by searching through the heap.

Multi-Robot Coordination

To implement this part, I began by creating the get_successors method which gets all the possible states from the state variable and then pass those onto a rules function that will check for the legality of those states and return those that pass. Then I finished up with the goal_test and the manhattan_heuristics function. I tested the get_successors within the Mazeworld Problem and then tested the A-star search with a MazeworldProblem in the test_mazeworld

1. For the state of the system of k robots, I would store it as a tuple with each x and y coordinate in the maze stored in the tuple. The state also needs to have a number for which robot to move so the total number of items in the state would be k*2 + 1
2. (n*n)*k
3. (w*k) + (n*n-w)*2 to estimate the collisions of robots you would take the number of walls, w, times the number of robots, k, and then to estimate how many collisions with other robots, take the total number of places without walls so n*n-w and multiply it by 2 because a robot can intersect with at least one of 2 other robots
4. I do not believe that computationally the breadth-first search would be able to handle such a large space due to the fact that it would have to store all the possible states that the search has gone through and that could be gigantic. Based off of the above equation you would get (100*100)*10 which is 100,000 possible states.
5. The heuristic I would use for this search space is manhattan heuristic so |dx + dy| from each robot to the goal and add all those up. This is monotonic because it might not always be a direct path because maybe a wall gets in the way and the robot may have to go around.
6. Maze4 is interesting because it forces the multiple robots to fit in narrow corridors and then make sure they align correctly. Maze5 is interesting because it takes multiple robots and while the heuristic may say they are closer to the goal than they actually are due to the walls in the way. Maze6 is interesting because there are only two areas for the robots to rearrange themselves in the beginning and at the end and I wanted to see which place it decided to do so. Maze7 is interesting in a similar way that Maze5 is except it is just 1 robot. Maze8 is interesting because the robots have to change from their starts in the corners to different corners and I wanted to see if they would cut throught the middle or go along the outside because if too many go along the outside then there is the possibility of them blocking each other. 7.The 8-problem is a special mazeworld problem because it is a maze with no walls and k = (n*n)-1. I do not believe the heuristic I chose to be optimal for this problem because just trying to move each robot to the goal would not be ideal because you might have to move a robot away from its goal to get another closer to its own goal so that the problem may be able

to be solved
7. I would modify the program to have a set of spots with robots and a set of spots without

Blind Robot

To implement the blind robot, I started with the move functions that I would use in the get_successors function to find the possible states from the current state. From there I created the goal_test function which tests if the state is a singleton and then decided to create a heuristic which figures out how far away from the point (0,0). Because the point is a corner it will more easily reach a singleton. The heuristic is very similar to the manhattan heuristic. The heuristic is optimistic but there probably are better heuristics.

Previous Work

In the work *Complete Algorithms for Cooperative Pathfinding Problems* by Trevor Standley and Richard Korf, the main problem they are facing is for real-world application to have an AI return an optimal path with that is also a quick solution to find. The result is they found an algorithm that returns a complete and optimal path with the drawbacks of an expensive run-time that might be too costly for any real world applications and that only conflicted paths are returned if the algorithm is terminated early. The basic approach of the paper is that they describe their solution and algorithm so that the reader understands how it works. They then compare their solution to other algorithms from other works.